Programmers Guide to GARA

March 2000

Contents

Note: Before you read about programming with GARA, you should have at least a passing familiarity with Globus. You can learn more about Globus at http://www.globus.org. This guide concentrates on describing GARA from a programmer's perspective. If you need information on installing and configuring GARA, please see the Administrators Guide to GARA. *If you would like more information about the research being done with GARA, please see the papers available at the Globus web site.*

About GARA

The GARA architecture provides programmers with convenient access to end-to-end quality of service (QoS) for programs. To do so, it provides mechanisms for making QoS reservations for different types of resources, including computers, networks, and disks. A reservation is a promise from GARA that an application will receive a certain level of service from a resource. For example, a reservation may promise a certain bandwidth on a network or a certain percentage of a CPU.

The GARA architecture is defined as a layered architecture with three levels of APIs and one level of low-level mechanisms:

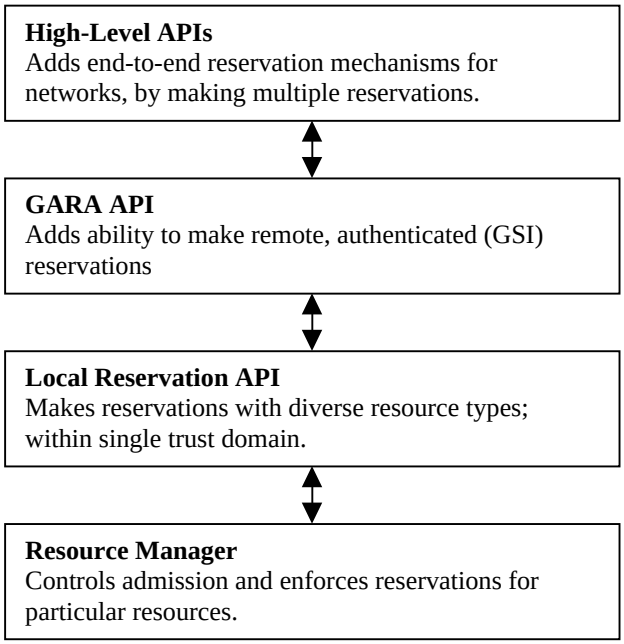

Figure 1 – GARA's layered architecture

Note that "GARA" refers to two things: the "GARA Architecture," which refers to the entire diagram above, and the "GARA API," which is the API for making a single reservation. This document describes just the GARA API; the other portions of GARA are described elsewhere.[1]

[1] At the time of this writing, these documents are still in preparation. However, many of the details can be learned from the GARA research papers that are available on the Globus web page: http://www.globus.org.

As a programmer, you will most likely be using the GARA API, and not the LRAM API, so it does not need to concern you further.

The GARA API has two interesting advantages. First, it allows you to make reservations either in advance of when you need them or right at the time that you need them—an *immediate reservation*. Second, you use the same API to make and manipulate a reservation regardless of the type of the underlying resource, thereby simplifying your programming when you need to work with multiple kinds of resources.

The GARA API can be considered a remote procedure call mechanism to communication with a resource manager. A resource manager controls reservations for a resource: it performs admission control and controls the resource to enforce the reservations. Some resources already have the ability to work with advanced reservations, so the resource manager is a simple program. Most resources cannot deal with advanced reservations, so the resource manager tracks the reservations and does admission control for new reservation requests. Much of the research in GARA has focused on building useful resource managers.

As you begin using the GARA API, you may discover that you need to make multiple simultaneous reservations. In the near future, high-level APIs to assist you with this sort of *co-reservation* task will be available.

Reservations

Reservations have five important attributes:

- *Start Time:* The time that the reservation begins. A reservation always has a start time, even if it is an immediate reservation, which begins as soon as you make the reservation. The start time is in seconds since 00:00:00 UTC, January 1, 1970. For example, if you want an immediate reservation, you can just call the Unix `time()` function.

- *Duration:* How long the reservation lasts, in second. All reservations must specify how long they will last, so that GARA can do appropriate admission control for reservations granted in advance.

- *Reservation Type:* The type of underlying resource, such as a network, a computer, or a disk.

- *Reservation Subtype*: A particular kind of reservation. See *Types of Network Reservations,* below.

- *Resource-Specific Parameters:* Parameters that are unique to each type of resource, such as bandwidth for a network reservation.

When you request a reservation, you specify these attributes. If your reservation request is accepted, you are provided with a *reservation handle.* This is an opaque string that uniquely identifies your reservation. All future operations require you to provide this handle.

Once you have a reservation handle, you can perform several operations with that handle:

- *Modify Reservation:* You can request a modification to your reservation. For instance, you can increase the bandwidth that you have requested.

- *Cancel Reservation:* You can inform GARA that you no longer need a reservation, by canceling it.

- *Claim Reservation:* When you are ready to use a reservation, you must claim the reservation. This is known as *binding* a reservation because you specify run-time parameters that you did not know when you created the reservation, such as ports being used for the network reservation.

- *Query Reservation:* You can discover the status of a reservation by polling it. The status includes whether the start of the reservation has begun and whether the reservation has been claimed.

- *Register Callback:* You can provide a function that will be called when the status of a reservation changes or when GARA wishes to provide extra information to your program. This information may include notification that your reservation appears to be too small. You can react to this information by modifying your reservation or changing your application behavior.

Types of Network Reservations

GARA implements several types of network reservations (see the reservation subtype above):

- *Foreground Reservations:* These are also called *normal* reservations. They are reservations for a specific bandwidth.
- *Background Reservations:* These are also called *bulk-transfer* reservations. A bulk transfer reservation shares all of the bandwidth not claimed by foreground reservations. The amount of bandwidth assigned to a particular reservation may change over time as foreground reservations begin and end. Programs are notified what the current bandwidth assignment is through a callback.
- *Low-Latency:* UDP flows that would like to avoid delays due to traffic shaping can request low-latency reservations.

Using GARA

GARA is provided as a library written in C. Any language that can link to C libraries can use GARA. There is also a Java implementation, but it is not described here. To use GARA, you will first need to have linked your program with these libraries. You will need to include "`globus_gara_client.h`" to provide prototypes for the GARA functions, and related constants. You will also need to include "`globus_common.h`" to gain access to `globus_module_activate()` and `globus_module_deactivate()`.

Initializing GARA

Before you can use GARA, you need to initialize it. GARA is initialized like other modules in Globus, using the `globus_module_activate`:

```
globus_module_activate(GLOBUS_GARA_CLIENT_MODULE);
```

Describing a Reservation Request

Reservation attributes are described using the Resource Specification Language (RSL). An RSL string is simply a list of attribute-value pairs that looks like

```
&(attribute-1=value-1) (attribute-2=value-2) É (attribute-N=value-N)
```

An example RSL string for requesting a network reservation for 150Kbps between looks like this:

```
&(reservation-type=network)
 (start-time=953158862)
 (duration=3600)
 (endpoint-a=140.221.48.146)
 (endpoint-b=140.221.48.106)
 (bandwidth=150)
```

Note that this string was spaced out on several lines for readability, while RSL strings do not have newlines in them.

Below is a list of attributes that may be used to specify a reservation. The universal attributes are for all types of reservations, while the other attributes are for specific types of resources. Note that the compute resource attributes are mutually exclusive, and currently only the percent-cpu attribute is used.

Attribute	Units	Default	Req?	Description
Universal Attributes				
reservation-type			Y	Allowable values: "network", "compute", or "disk"
reservation-subtype				Currently valid only for network reservations. If it is not specified, it is a foreground reservation. Otherwise it is one of "background" or "low-latency". For more information, see *Types of Network Reservations* above.
start-time	secs		Y	What time the reservation starts in seconds since 00:00:00 UTC, January 1, 1970. If you specify "now", then the reservation will begin immediately.
duration	secs	100		Length of the reservation, in seconds.
Compute Resource Attributes				
percent-cpu	%	20		Percentage of the CPU's time given to the reserved process.
Network Resource Attributes				
endpoint-a			Y	The machine at one end of the network flow. This must be specified as a dotted IP address, such as 140.221.48.162.
endpoint-b			Y	The machine at the other end of the network flow. This must be specified as a dotted IP address, such as 140.221.48.162.
bandwidth	Kbps	8		How fast a flow can transfer data.
directionality**		bidirectional		`unidirectional-ab`: reservation for traffic from a to b. `unidirectional-ba`: reservation for traffic from b to a. `bidirectional`: reservation for traffic in both directions.
Disk Resource Attributes				
size	KB			The storage space needed for a single file.
bandwidth	Kbps	8		How fast data can be read/written to a file.

** directionality is ignored in the current (March 2000) version of GARA. Right now, "unidirectional-ab" is assumed.

Creating a Reservation

Before you can create a reservation, you will need to specify your reservation. See *Describing your Reservation Request* above. Then you request your reservation with globus_gara_reservation_create (spacing adjusted for clarity):

Before you can create a reservation, you will need to specify your reservation. See *Describing your Reservation Request* above. Then you request your reservation with globus_gara_reservation_create (spacing adjusted for clarity):

```
int  error;
char *request_rsl = Ò&(reservation-type=network)
                    (start-time=953158862) (duration=3600)
                    (endpoint-a=140.221.48.146)
                    (endpoint-b=140.221.48.106)
                    (bandwidth=150)Ó;
char *reservation_handle;

error = globus_gara_reservation_create(gatekeeper_contact, request_rsl,
                                       &reservation_handle);
```

Note that the gatekeeper contact is a string obtained from another location, such as the MDS. An example gatekeeper contact may look like

dslnet2.mcs.anl.gov:754:/C=US/O=Globus/O=Argonne National Laboratory/OU=Mathematics and Computer Science Division/CN=dslnet2.mcs.anl.gov

For more information on gatekeeper and gatekeeper contacts, see http://www.globus.org, and read about GRAM.

Modifying a Reservation

Modifying a reservation is similar to creating a reservation, except that instead of providing a gatekeeper contact, you provide the handle to the reservation that you created earlier:

```
int  error;
char *request_rsl = Ò&(reservation-type=network)
                     (start-time=953158862) (duration=7200)
                     (endpoint-a=140.221.48.146)
                     (endpoint-b=140.221.48.106)
                     (bandwidth=200)Ó;

error = globus_gara_reservation_create(reservation_handle, request_rsl,
                                &reservation_handle);
```

Querying a Reservation

If you would like to find out the status of a reservation, you can query it:

```
int error;
int status;

error = globus_gara_reservation_status(reservation_handle, &status);
```

If there is not an error, the status will be one of

GLOBUS_GARA_RESERVATION_STATUS_NOT_STARTED
GLOBUS_GARA_RESERVATION_STATUS_NOT_STARTED_BOUND
GLOBUS_GARA_RESERVATION_STATUS_READY_NOT_BOUND
GLOBUS_GARA_RESERVATION_STATUS_ACTIVE
GLOBUS_GARA_RESERVATION_STATUS_FINISHED

A reservation is bound if a previous call to globus_gara_reservation_bind succeeded. A reservation is ready if the current time is later than the start time, and the duration has not yet elapsed. A reservation is active if it is both ready and bound. A reservation is finished if the current time is later than the start time plus the duration.

Binding a Reservation

When you are ready to use a reservation, you need to *bind* it in order to begin using the reservation:

```
int   error;
char  *bind_paramters = Ò&(which-endpoint=a)(endpoint-a-port=1234)
                         (endpoint-b-port=5678)Ó;

error = globus_gara_reservation_bind(reservation_handle,
           &bind_parameters);
```

Notice that the run-time parameters are specified as an RSL string. Currently, bind parameters are only specified for compute and network reservations. For compute reservations, the only parameter to be specified is `process-id`, which specifies the process ID of the process that will be receiving the reservation.[2] For network reservations, there are three parameters:

- `which-endpoint`: If the reservation is being bound from a machine involved in the reservation, this specifies which machine it is. The machine is either Òa Óor Òb Ó, and it matches what was specified in the reservation request. If a different machine is binding the reservation on behalf of the processes involved, simply use Òa Ó

[2] Note that the process ID is relevant only if you are using the DSRT resource manager to control scheduling for processes with reservations.

- `endpoint-a-port`: This is the port used by endpoint-a, as specified in the reservation request. Because the current GARA implementation assumes that data is being sent from endpoint-a to endpoint-b, this will be the port used by the sender.
- `endpoint-b-port`: This is the port use by endpoint-b, as specified in the reservation request. Because the current GARA implementation assumes that data is being sent from endpoint-a to endpoint-b, this will be the port used by the receiver.

Note that a reservation is not considered active until it is bound. Once a reservation has both begun and been bound, the GARA do whatever setup is necessary in order to ensure that the reservation is granted. It is okay if the reservation is bound before it has begun, because GARA will automatically enable the reservation once it begins.

If you will temporarily not be using a reservation but you will resume using it before it has expired, you can unbind the reservation:

```
int   error;

error = globus_gara_reservation_unbind(reservation_handle);
```

Once you unbind a reservation, you may bind it again.

Using Callbacks

If you would like to be informed whenever the status of a reservation changes (see *Querying a Reservation* above), you can use a callback function. Once you register a callback function, it will immediately be called once, to provide the current status, and will be called every time the status changes afterwards.

First you need to create a callback function:

```
static void callback_handler(
    char                              *reservation_handle,
    globus_gara_reservation_event_t  event,
    void                              *user_parameter)
{
  /* Place code here to examine the event */
  /* If it is a status event, event.event_type will be
     GLOBUS_GARA_STATUS_EVENT, and the status will be in
     event.event. */
  if (event.event == GLOBUS_GARA_STATUS_EVENT)
  {
    if (event.event_type == GLOBUS_GARA_RESERVATION_STATUS_FINISHED)
    {
      /* React to reservation being finished */
    }
  }
  return;
}
```

Then you need to register this function with GARA. You need to register the function for each reservation that you wish to monitor:

```
int error;

error = globus_gara_reservation_callback_register(reservation_handle,
            callback_handler, NULL);
```

Note that the last parameter you pass to the registration function will be forwarded as the `user_parameter` to your callback function.

If you would no longer like to have a function called when the status changes, you can unregister it:

```
int error;

error = globus_gara_reservation_callback_remove(reservation_handle,
            callback_handler);
```

Note you can register multiple callback functions for a single reservation handle.

Canceling a Reservation

When you are all done using a reservation, you should cancel it, using the reservation handle that you obtained when you created the reservation.

```
globus_gara_reservation_cancel(reservation_handle);
```

When you cancel a reservation, all of the callbacks that have been registered for that reservation will automatically be cancelled.

Deactivating GARA

When you have finished using GARA, you should deactivate it, to allow it to clean up:

```
globus_module_deactivate(GLOBUS_GARA_CLIENT_MODULE);
```

GARA Reference

Constants

This section describes the constants used by the GARA API. You will find them all either in "globus_gara_client.h" or in "globus_gara_common.h". Note, however, that "globus_gara_client.h" includes "globus_gara_common.h" for you.

Errors

GLOBUS_GARA_ERROR_NONE
: No error has occurred.

GLOBUS_GARA_ERROR_UNKNOWN
: An error has occurred, but GARA just doesn't know what it is.

GLOBUS_GARA_ERROR_MODULE_NOT_ACTIVE
: You have tried to use GARA without activating the module first.

GLOBUS_GARA_ERROR_BAD_PARAMETER
: A bad parameter, such as a NULL reservation handle, has been passed to a GARA function.

GLOBUS_GARA_ERROR_ZERO_LENGTH_RSL,
: An RSL string was provided, but it is empty. It may be that this is never returned.

GLOBUS_GARA_ERROR_BAD_RSL,
: There is an error, probably a syntax error, in the RSL string.

GLOBUS_GARA_ERROR_BAD_RESERVATION_HANDLE
: The reservation handle that was provided isn't really a reservation handle.

GLOBUS_GARA_ERROR_CONNECTION_FAILED
: GARA was unable to connect to the gatekeeper.

GLOBUS_GARA_ERROR_AUTHORIZATION
: GARA was unable to authorize with the gatekeeper. Did you run grid-proxy-init?

GLOBUS_GARA_ERROR_GATEKEEPER_MISCONFIGURED
: I don't think this is ever reported, so it probably doesn't mean anything.

GLOBUS_GARA_ERROR_VERSION_MISMATCH
: I don't think this is ever reported, so it probably doesn't mean anything.

GLOBUS_GARA_ERROR_INVALID_REQUEST
: I don't think this is ever reported, so it probably doesn't mean anything.

GLOBUS_GARA_ERROR_UNKNOWN_RESERVATION_TYPE
: The reservation type in the RSL reservation request must be one of "network", "compute", or "disk", but it wasn't.

GLOBUS_GARA_ERROR_PROTOCOL_FAILED
: There was a problem communicating with the gatekeeper.

GLOBUS_GARA_ERROR_MISSING_RESERVATION_TYPE
: The reservation type in the RSL reservation request wasn't provided.

GLOBUS_GARA_ERROR_OUT_OF_MEMORY
: A request for memory failed. You're in trouble!

GLOBUS_GARA_ERROR_MISSING_ENDPOINT_A
: A network reservation request didn't specify endpoint-a.

GLOBUS_GARA_ERROR_MISSING_ENDPOINT_B
: A network reservation request didn't specify endpoint-b.

GLOBUS_GARA_ERROR_CANT_MAKE_RESERVATION

The reservation can't be made. Probably there are other reservations already at the same time, and there isn't room for your reservation.

GLOBUS_GARA_ERROR_PROBLEM_WITH_LRAM
: The most likely cause of this error is that the resource manager is not running or that communication with it has failed.

GLOBUS_GARA_ERROR_HTTP_UNPACK_FAILED
: A serious protocol error happened, probably a programming error on our part, not yours.

GLOBUS_GARA_ERROR_BAD_RESERVATION_OBJECT
: This error probably means that you tried to make a network reservation for an endpoint that the resource manager hasn't been configured to allow reservations for.

GLOBUS_GARA_ERROR_GARA_SERVICE_EXECUTABLE_NOT_FOUND
: The gatekeeper is misconfigured. In particular, it can't find the globus_gatekeeper_gara_service executable.

GLOBUS_GARA_ERROR_CANT_CONTACT_RESOURCE_MANAGER
: The resource manager is unavailable. Check to make sure that it's running.

GLOBUS_GARA_ERROR_UNKNOWN_GRAM_ERROR
: Some error in the underlying "GRAM Gatekeeper" protocol has failed.

GLOBUS_GARA_ERROR_MISSING_RESERVATION_SUBTYPE
: I don't think this is ever reported, so it probably doesn't mean anything.

Callback and Status Constants

The following events are reported to callbacks:

GLOBUS_GARA_STATUS_EVENT
: The status of the reservation has changed. See the lists of status constants below.

GLOBUS_GARA_CHANGE_EVENT
: The reservation has been preempted, or the reservation quantity (like bandwidth) has changed. See the list changes below.

GLOBUS_GARA_MONITOR_EVENT
: Not yet used.

The following statuses can be reported to callbacks on a status event or in response to a user calling `globus_gara_reservation_status`.

GLOBUS_GARA_RESERVATION_STATUS_NOT_STARTED
: The reservation has not yet begun (the current time is before the start time).

GLOBUS_GARA_RESERVATION_STATUS_NOT_STARTED_BOUND
: Although the reservation has not yet begun, the reservation has been bound.

GLOBUS_GARA_RESERVATION_STATUS_READY_NOT_BOUND
: The reservation has begun (the current time is after the start time) but can't yet be used because it has not been bound yet.

GLOBUS_GARA_RESERVATION_STATUS_ACTIVE
: The reservation has begun and been bound.

GLOBUS_GARA_RESERVATION_STATUS_FINISHED

The reservation is over. That is, the current time is greater than the start time plus the duration of the reservation.

The following changes can be reported on a CHANGE_EVENT:

GLOBUS_GARA_RESERVATION_CHANGE_PREEMPTED
: The reservation has been preempted because a more important reservation has occurred. Currently, this will not be reported, because preemption has not yet been implemented.

GLOBUS_GARA_RESERVATION_CHANGE_QUANTITY
: The quantity (like bandwidth) has been changed. This occurs for bulk transfer reservations.

Data Structures

This is a description of the data structures used by GARA.

The Event Data Structure

```
typedef struct
{
  int     event_type;
  int     event;
  double  quantity;
} globus_gara_reservation_event_t;
```

This structure is provided to callback functions. The event type and event are constants from the list above. The quantity is provided when the event is a change event indicating that the quantity has changed.

Callback functions

```
typedef void (*globus_gara_reservation_callback_t)(
    char                            *reservation_handle,
    globus_gara_reservation_event_t  event,
    void                            *user_parameter);
```

This is the type of function that must be used for callback functions
.

Functions

Note that all of the functions in GARA return an integer. This integer is the error code, if any error occurred. See the list of errors under *Constants*.

globus_gara_reservation_create

```
int globus_gara_reservation_create(
          const char *manager_contact,
```

```
          const char *reservation_specification,
          char       **reservation_handle);
```

This function attempts to make a reservation.

In:

`manager_contact`: The contact string for the gatekeeper that controls access to the resource manager for the resource you wish to make a reservation with.

`reservation_specification`: An RSL string describing the attributes you wish to have for your reservation. See *Describing a Reservation Request* above.

Out:

`reservation_handle`: If the reservation was successfully made, a pointer to your reservation handle will be provided in this parameter. The memory for the reservation handle is allocated by `globus_malloc()`, and it is your responsibility to free the memory with `globus_free()` when you are done.

globus_gara_reservation_modify

```
int globus_gara_reservation_modify(
          const char *old_reservation_handle,
          const char *reservation_specification,
          char       **new_reservation_handle);
```

This function attempts to modify a new reservation. Note that if the reservation is changed, you are provided with a new reservation handle. While current versions of GARA will provide an identical reservation handle, future versions of GARA may not.

In:

`old_reservation_handle`: The handle for the reservation that you wish to modify.

`reservation_specification`: An RSL string describing the new attributes you wish to have for your reservation. See *Describing a Reservation Request* above.

Out:

`new_reservation_handle`: If the reservation was successfully modified, a pointer to your reservation handle will be provided in this parameter. The memory for the reservation handle is allocated by `globus_malloc()`, and it is your responsibility to free the memory with `globus_free()` when you are done.

globus_gara_reservation_bind

```
int globus_gara_reservation_bind(
          const char *reservation_handle,
          const char *bind_parameters);
```

This claims a reservation by providing run-time parameters.

In:

`reservation_handle`: The handle for the reservation that you wish to bind.

`bind_parameters`: An RSL string describing the new attributes you wish to have for your reservation. See *Binding a Reservation* above.

globus_gara_reservation_unbind

```
int globus_gara_reservation_unbind(
          const char *reservation_handle);
```

This ÒUn-claimsÓ a reservation. The reservation is still valid and can be used again by calling `globus_gara_reservation_bind()` again.

In:

`reservation_handle`: The handle for the reservation that you wish to bind.

globus_gara_reservation_status

```
int globus_gara_reservation_status(
          const char *reservation_handle,
          int        *status;
```

This function queries for a reservationÕs status.

In:

`reservation_handle`: The handle for the reservation that you wish to query.

Out:

`status`: The status of the reservation. It is one of the constants described in *Callback and Status Constants.*

globus_gara_reservation_callback_register

```
int globus_gara_reservation_callback_register(
          const char                          *reservation_handle,
          globus_gara_reservation_callback_t  callback_function,
          void                                *user_parameter);
```

After this function successfully completes, the specified callback function will be called whenever the status of a reservation changes. It will also be immediately called once to provide the current status of the reservation. Note that multiple callbacks can be registered for a single reservation.

In:

`reservation_handle`: The handle for the reservation for which you wish to receive callbacks.

`callback_function`: The function that will be called by GARA when the status of a reservation changes.

`user_parameter`: The value you provide here will be provided to the callback function unmodified.

globus_gara_reservation_callback_remove

```
int globus_gara_reservation_callback_remove(
          const char                          *reservation_handle,
          globus_gara_reservation_callback_t  callback_function);
```

After this function successfully completes, the specified callback function will no longer be called when the status of the reservation changes.

In:

`reservation_handle`: The handle for the reservation for which you wish to receive callbacks.

`callback_function`: The function that will be called by GARA when the status of a reservation changes.

`user_parameter`: The value you provide here will be provided to the callback function unmodified.

globus_gara_reservation_cancel

```
int globus_gara_reservation_cancel(
          const char *reservation_handle);
```

This cancels a reservation. When a reservation is cancelled, the reservation handle (and copies of it) may not be used anymore. For example, if you try to bind the cancelled reservation, it will fail.

In:

`reservation_handle`: The handle for the reservation that you wish to cancel.

globus_gara_reservation_version

```
int globus_gara_version(void);
```

This returns the current version number for GARA.

globus_gara_reservation_client_debug

```
int globus_gara_client_debug(void);
```

This enables debugging mode. Output will be printed to stderr.

In:

`reservation_handle`: The handle for the reservation that you wish to cancel.

globus_gara_client_error_string

```
const char *globus_gara_client_error_string(
          int error_code);
```

For any error code returned by GARA, this provides a printable string that corresponds to the error code.

In:

`error_code`: The error code for which you wish to obtain a string representation.

Appendix: Example Program Using GARA

This code has been greatly simplified but should give you the basic idea of how to use GARA.

```
#include "globus_gara_client.h"

int reservation_ready = GLOBUS_FALSE;

static void callback_handler(
    char                              *reservation_handle,
    globus_gara_reservation_event_t  event,
    void                              *user_parameter);

int main(int argc, char **argv)
{
  int           error;
  int           seconds;
  int           test_index;
  char          *reservation_handle;
  char          *reservation_rsl_string;

  /* Initialize */
  globus_module_activate(GLOBUS_GARA_CLIENT_MODULE);

  /* Make a reservation that starts in 20 seconds and goes for an hour */
  /* Of course, many of the parameters would not actually be static in a
   * real program. */
  reservation_rsl_string = globus_malloc(256);
  sprintf(reservation_rsl_string,
          Ò"&(reservation-type=network)(start-time=%d)(duration=%d) \
(endpoint-a=%s) (endpoint-b=%s) (bandwidth=%d) (protocol=tcp)",
            (int) time(NULL) + 20, 3600,
            Ò128.135.11.1Ó, Ò128.135.11.6Ó, 150);
  error = globus_gara_reservation_create(parameters.gatekeeper_contact,
              reservation_rsl_string, &reservation_handle);

  /* Set up a callback to let us know when the reservation is ready. */
  error = globus_gara_reservation_callback_register(reservation_handle,
              callback_handler, NULL);

  /* Wait for the reservation to become active */
  while (!reservation_ready)
    sleep(1);

  /* Bind the reservation */
  error = globus_gara_reservation_bind(reservation_handle,
              "&(which-endpoint=a) (endpoint-a-port=9999) \
(endpoint-b-port=9999)");

  /* Use the reservation... */
  ;

  /* Remove the callback */
```

```
  error = globus_gara_reservation_callback_remove(reservation_handle,
              callback_handler);

  /* Cancel the reservation */
  error = globus_gara_reservation_cancel(reservation_handle);

  /* Clean up */
  globus_free(reservation_rsl_string);
  globus_free(reservation_handle);
  globus_module_deactivate(GLOBUS_GARA_CLIENT_MODULE);
  return 0;
}

static void callback_handler(
    char                              *reservation_handle,
    globus_gara_reservation_event_t  event,
    void                              *user_parameter)
{
  if (   event.event_type == GLOBUS_GARA_STATUS_EVENT
      && event.event      == GLOBUS_GARA_RESERVATION_STATUS_READY_NOT_BOUND)
  {
    reservation_ready = GLOBUS_TRUE;
  }
}
```